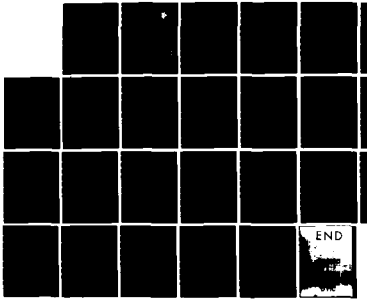
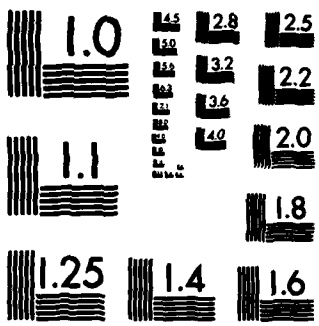
AD-A144 244 LISP IMPLEMENTATION BASELINE INVESTIGATION(U) ROME AIR DEVELOPMENT CENTER GRIFFISS AFB NY R C SCHRAG JUN 84 RADC-TR-84-151 1/1

UNCLASSIFIED F/G 9/2 NL






MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

RADC-TR-84-151
In-House Report
June 1984

# LISP IMPLEMENTATION BASELINE INVESTIGATION

Robert C. Schrag

AD-A144 244

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
AUG 9 1984
S
B

DTIC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

84 08 10 008

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-84-151 has been reviewed and is approved for publication.

APPROVED: Samuel A. D. Mitto

SAMUEL A. DINITTO, JR.
Chief, Command & Control
Software Technology Branch
Command & Control Division

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Command & Control Division

FOR THE COMMANDER: Donald A. Brantingham

DONALD A. BRANTINGHAM
Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS: N/A |
| 2a. SECURITY CLASSIFICATION AUTHORITY: N/A | 3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited. |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S): RADC-TR-84-151 | 5. MONITORING ORGANIZATION REPORT NUMBER(S): N/A |
| 6a. NAME OF PERFORMING ORGANIZATION: Rome Air Development Center; 6b. OFFICE SYMBOL (If applicable): COES | 7a. NAME OF MONITORING ORGANIZATION: N/A |
| 6c. ADDRESS (City, State and ZIP Code): Griffiss AFB NY 13441 | 7b. ADDRESS (City, State and ZIP Code): |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center; 8b. OFFICE SYMBOL (If applicable): COES | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N/A |
| 8c. ADDRESS (City, State and ZIP Code): Griffiss AFB NY 13441 | 10. SOURCE OF FUNDING NOS. |

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT NO. |
|---|---|---|---|
| 62702F | 5581 | 19 | 12 |

11. TITLE (Include Security Classification): LISP IMPLEMENTATION BASELINE INVESTIGATION

12. PERSONAL AUTHOR(S): Robert C. Schrag

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Yr., Mo., Day) | 15. PAGE COUNT |
|---|---|---|---|
| In-House | FROM Apr 80 TO Apr 84 | June 1984 | 48 |

16. SUPPLEMENTARY NOTATION:

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB. GR. | LogLisp, expert systems, logic programming, pattern matching, artificial intelligence |
| 9 | 2 | 14 | |
| 9 | 2 | 15 | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

LogLisp is an Artificial Intelligence (AI) programming environment that fully combines the facilities of logic programming and Lisp. This report describes the implementation of a simple knowledge-based system in LogLisp, performed as part of an effort to evaluate the effectiveness of LogLisp for implementing a simple knowledge-based system--in terms of programming ease, program clarity, and program efficiency--using its original Lisp implementation as a baseline.

For the baseline investigation we chose the knowledge-based system MicroKnobs, a prototype tactical air mission planning system whose chief function is to select munitions for a known target based on rules and facts about targets, target conditions, and munitions. Our implementation preserves the outward behavior and control features of MicroKnobs while replacing the original Lisp-coded knowledge base and inference system, pattern matcher, and dictionary with LogLisp-coded counterparts. (cont'd)

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐ | UNCLASSIFIED |

| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE NUMBER (Include Area Code) | 22c. OFFICE SYMBOL |
|---|---|---|
| Robert C. Schrag | (315) 330-2748 | RADC (COES) |

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

The report includes descriptions of MicroKnobs interaction and of the software architectures of its Lisp and our LogLisp implementations, and an evaluation of our implementation and selected design alternatives against the original MicroKnobs in terms of programming ease, program clarity, and program efficiency. The results indicate that LogLisp is a viable environment for the implementation and development of knowledge-based systems.

Acknowledgements

Nort Fowler conceived and initiated this effort. Sharon Walter helped me to perform the LogLisp implementation of MicroKnobs. Phil Prince reviewed this report. Thank you all.

Carl Engelman, to whose memory this report is dedicated, was always gracious in providing documentation, advice, and encouragement.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist Special
A-1

# 1. Lisp Implementation Baseline Investigation

This report describes the implementation of a simple knowledge-based system in LogLisp, performed as part of a Rome Air Development Center (RADC) in-house effort to experiment with and evaluate LogLisp. The objective of this task has been to evaluate the effectiveness of LogLisp for implementing a simple knowledge-based system in terms of programming ease, program clarity, and program efficiency, using its original Lisp implementation as a baseline.

## 1.1. LogLisp

LogLisp (logic programming in Lisp) is an Artificial Intelligence (AI) programming environment developed by Syracuse University under contract to RADC [Robinson and Sibert 81]. In it coexist the tools of two competing philosophies of AI programming--functional programming done in Lisp, and logic programming. Each language has characteristics which are appropriate for encoding major AI program components including knowledge representation, knowledge manipulation, and inference.

Lisp (list processing language) facilitates knowledge representation and manipulation since almost any formal knowledge representation can be expressed as list structure and manipulated via Lisp primitives. Lisp includes no built-in inference mechanisms, and these are often reprogrammed with each new application and accompanying knowledge representation. (For tutorial information on Lisp see [Winston and Horn 80.])

Logic programming languages include a built-in theorem-prover that is responsible for all program execution. Statements (or clauses) are expressed in a subset of predicate logic. Most formal knowledge representations can be expressed in clausal form. Knowledge manipulation can be awkward in pure logic programming, however, since it includes neither an explicit assignment statement nor extensive data manipulation primitives. (See [Kowalski 79] for background information on logic programming.)

LogLisp promotes synthesis of these two very complementary programming styles. While a user can disregard either the logic programming component (hereinafter called Logic) or the underlying Lisp system, Lisp can be used to call Logic and vice versa, to arbitrary levels. LogLisp augments Lisp with an inference engine of tested capability. Logic programming is enhanced by the opportunity to call Lisp for knowledge manipulation and by Lisp's more developed support environment.

## 1.2. MicroKnobs

For the baseline investigation we chose the knowledge-based system MicroKnobs [Engelman 79], a prototype tactical air mission planning system developed by MITRE Corporation to demonstrate the applicability of knowledge-based system technology to Air Force problems. The demonstration knowledge base contains only ten rules and eight facts about targets, target conditions, and munitions. Its chief function is to select the best munitions for a known target, based on the contents of its knowledge base. Different, consistent results are obtained after on-line modification of the knowledge base, showing the power of the knowledge-based system approach. MicroKnobs has been succeeded by the MITRE Knobs (Knowledge-based System) project funded by RADC [Engelman 81].

MicroKnobs has several features which suit it to the baseline study. It is rule-based. Rules of the form used in the knowledge base of MicroKnobs are frequently called production rules. These take the form: IF A1 & A2 & ... & An THEN B1 & B2 & ... & Bm, where n >= 0 and m >= 1. Production rules are easily translated to the form used in logic programming knowledge bases:
B IF A1 & A2 & ... & An, where n >= 0. This is done by applying the distributive law to the conjunction in the conclusion of a production rule with multiple consequents to form m distinct logic programming clauses, each with the original (identical) antecedents.

MicroKnobs includes a custom user-interface which must be preserved in any faithful reimplementation. It includes custom inference, pattern-matching, and control mechanisms. We succeeding in preserving the outward behavior and control features of MicroKnobs while replacing portions of the original Lisp-based implementing machinery with Logic-based machinery, as the overall MicroKnobs system architecture and our knowledge of LogLisp programming techniques suggested.

MicroKnobs is witten in InterLisp [Teitelman 78]. As a preliminary task we implemented LogLisp V2M3 in InterLisp [Schrag 83]. This allows us to perform direct comparisons.

## 1.3. Organization of this Report

The remainder of this report is comprised of four major sections. Section 2 describes the MicroKnobs system from the viewpoint of user interaction. Section 3 describes the software architecture of the MITRE implementation of MicroKnobs. In section 4 we describe our LogLisp

implementation of MicroKnobs, emphasizing design development and rationale. In section 5 we evaluate our implementation and selected design alternatives against the original MicroKnobs in terms of programming ease, program clarity, and program efficiency. A final section lists references.

## 2. Interaction with MicroKnobs

The user interacts with MicroKnobs via a monitor which interprets natural language input to invoke the various system functions. He may examine the knowledge base by asking to see either facts or rules, including target rules (TRULES) and munitions rules (MRULES). (In the following scripts, "&&" is the monitor's prompt. The entire knowledge base is shown here. Some representative facts and rules are marked (***) and will be referred to in subsequent discussion.)

```
        WELCOME TO KNOBS
        TYPE HELP FOR INSTRUCTIONS

        && PRINT FACTS

***     1:  THE CLOUD COVER OF TARGET 4 IS 5

***     2:  TARGET 1 IS A BRIDGE

        3:  TARGET 2 IS A TANK BATTALION

        4:  TARGET 3 IS AN SA-4

***     5:  TARGET 1 IS LIGHTLY DEFENDED

        6:  THE CLOUD COVER OF TARGET 1 IS 0

        7:  THE CLOUD COVER OF TARGET 2 IS 3

        8:  TARGET 4 IS AN AIRBASE
```

Interaction with MicroKnobs

```
       && PRINT RULES

                                        --TRULES--

                                        --TR1--
***    IF:
***            1:  THE target IS ONE OF: RADAR SAM RADIO
***
***    THEN:
***            1:  THE target IS AN EMITTER

                                        --TR2--
***    IF:
***            1:  THE target IS A TANK BATTALION
***
***    THEN:
***            1:  THE target IS FAST
***       AND  2:  THE target IS WARM

                                        --TR3--
       IF:
              1:  THE target IS A BRIDGE

       THEN:
              1:  THE target IS COLD
          AND 2:  THE target IS VERY HARD
          AND 3:  THE target IS STATIONARY

                                        --TR4--
       IF:
              1:  THE target IS ONE OF: SA-2 SA-4 SA-6

       THEN:
              1:  THE target IS A SAM

                                        --TR5--
       IF:
              1:  THE target IS AN AIRBASE

       THEN:
              1:  THE target IS HARD
          AND 2:  THE target IS HIGHLY DEFENDED

                          -- END OF TRULES --
```

Interaction with MicroKnobs

```
                                    --MRULES--

                                     --MR1--
***     IF:
***             1:  THE target IS HARDER THAN HARD
***        AND  2:  THE CLOUD COVER OF THE target IS LESS THAN 1
***        AND  3:  THE target IS LESS DEFENDED THAN LIGHTLY DEFENDED
***
***     THEN:
***             1:  THE BEST MUNITIONS FOR THE target IS LASER GUIDED

                                     --MR2--
        IF:
                1:  THE target IS SOFTER THAN SOFT

        THEN:
                1:  THE BEST MUNITIONS FOR THE target IS CLUSTER BOMBS

                                     --MR3--
        IF:
                1:  THE target IS FASTER THAN FAST
           AND  2:  THE target IS WARMER THAN WARM

        THEN:
                1:  THE BEST MUNITIONS FOR THE target IS SHILLELAGH

                                     --MR4--
        IF:
                1:  THE target IS AN EMITTER

        THEN:
                1:  THE BEST MUNITIONS FOR THE target IS SHRIKE

                                     --MR5--
        IF:
                1:  THE target IS SOFTER THAN HARD
           AND  2:  THE CLOUD COVER OF THE target IS LESS THAN 4
           AND  3:  THE target IS MORE DEFENDED THAN HIGHLY DEFENDED

        THEN:
                1:  THE BEST MUNITIONS FOR THE target IS MAVERICK


                            -- END OF MRULES --
```

Interaction with MicroKnobs

The user may invoke inference by asking MicroKnobs to choose munitions for a certain target.

```
&& CHOOSE MUNITIONS FOR TARGET 1
LASER GUIDED
```

He may request an explanation of the previous choice.

```
&& WHY
BY MR1: THE BEST MUNITIONS FOR TARGET 1 IS LASER GUIDED
   SINCE:
       1: BY TR3: TARGET 1 IS VERY HARD
             SINCE:
                 1: DATA:  TARGET 1 IS A BRIDGE
       2: DATA:  THE CLOUD COVER OF TARGET 1 IS 0
       3: DATA:  TARGET 1 IS LIGHTLY DEFENDED
```

Interaction with MicroKnobs

The user may also edit the knowledge base by adding or deleting facts or rules, and by modfiying or moving rule components. The editor accepts rule components expressed in natural language.

```
&& EDIT TR2
                                --TR2--
IF:
        1:  THE target IS A TANK BATTALION

THEN:
        1:  THE target IS FAST
   AND  2:  THE target IS WARM

CHANGE NAME?   NO

EDIT HYPOTHESES?   NO

EDIT CONSEQUENCES?   YES

ADD, DELETE, OR REPLACE?   ADD

CONSEQUENCE NUMBER:   3

CONSEQUENCE NUMBER 3 :   THE TARGET IS HARD.

CONSEQUENCES:
        1:  THE target IS FAST
   AND  2:  THE target IS WARM
   AND  3:  THE target IS HARD

EDIT CONSEQUENCES?   NO

                                --TR2--
IF:
        1:  THE target IS A TANK BATTALION

THEN:
        1:  THE target IS FAST
   AND  2:  THE target IS WARM
   AND  3:  THE target IS HARD
```

In addition to these commands, the user may ask questions about the knowledge base. He can ask for information about rules.

```
&& WHAT RULES MENTION HARDNESS IN THE CONCLUSION?
THE FOLLOWING RULES MENTION HARDNESS :
        TR3 TR5
```

User queries can invoke inference to deduce information contained in the knowledge base implicitly.

```
&& HOW HARD IS TARGET 4?
HARD

&& WHY?
BY TR5: TARGET 4 IS HARD
   SINCE:
       1: DATA:  TARGET 4 IS AN AIRBASE
```

## 3. Original MicroKnobs System Architecture

MicroKnobs has a modular architecture. Its major components are the monitor, parler/dictionary, pattern-matching interpreter and associated pattern/production rules, knowledge base, knowledge base display package, inference engine, and knowledge base editor.

### 3.1. Parler/Dictionary

The monitor is the central point, if not the heart, of MicroKnobs. It is the dispatcher and receiver of control to and from other system modules. All natural language input (including that to the monitor) is first passed through the parler. (In contrast to a parser, it uses no grammar.) The parler accesses the dictionary of single- and multiple-word terms known by MicroKnobs. Punctuation is stripped from input, preferred terms are substituted for synonyms, and any multiple-word terms are standardized (made into single Lisp atoms by hyphenating). Parling helps to ensure that user input is consistent with language in the knowledge base. This is important because exact match is used in inference.

The dictionary is organized around a list of preferred terms called SYNONYMWORDS. Each entry on this list has stored under the property SYNONYM on its property list a list of the unpreferred terms that map into it. The SYNONYM property value for the term AIRBASE and the mapping function GETROOTWORD are shown below.

```
(AIRFIELD AIRPORT TUOC (AIR BASE) (AIR FIELD))

(GETROOTWORD
  [LAMBDA (WORD)
    (FOR EACHROOT IN SYNONYMWORDS
       DO (COND
            ([OR (EQUAL EACHROOT WORD)
                 (MEMBER WORD (GETPROP EACHROOT (QUOTE SYNONYM]
              (RETURN EACHROOT])
```

GETROOTWORD performs sequential search over the entire dictionary.

### 3.2. Pattern-matching System

Parled monitor input is compared by the pattern-matching interpreter against two sets of pattern/production rules, successively. The pattern/production rules are like conventional production rules, with some exceptions. They are executed singly, for effect, rather than chained together in inference. They are appropriately considered in terms of condition:action rather than hypothesis:conclusion. Their

condition parts are of unit length and are expressed in a pattern-matching language rather than predicate logic. The pattern-matching language includes variables to match input patterns of unit length (atoms whose first character is "?") and variable length (atoms whose first character is "!"), and a variable-length-don't-care symbol ("!"). Unit- and variable-length (but not anonymous) variables can also appear in the action parts of pattern/production rules, and are then subject to instantiation using the matching substitutions created by the replacement of terms for variables in the condition parts. The condition part of a pattern/production rule can include constraints, which are calls to Lisp functions with instantiated pattern-matching variables as arguments. The constraints must evaluate to other than NIL in order for the condition to be satisfied. As example, the pattern/production rule which fires when the query, "WHICH RULES MENTION HARDNESS IN THE CONCLUSION?", (demonstrated in the previous section) is submitted to the monitor is shown below.

```
                                                    {(comments)}
[[(?INTEROG ! ?ATTRIB ! ?SPECATTRIB ! ?INFO)        {the pattern}
  [?INTEROG (OR (EQ ?INTEROG (QUOTE WHAT))          {constraint on
                (EQ ?INTEROG (QUOTE WHICH]                  ?INTEROG}
  [?ATTRIB (OR (EQ ?ATTRIB (QUOTE CHOOSE))          {...on ?ATTRIB}
               (EQ ?ATTRIB (QUOTE MENTION]
  (?SPECATTRIB (OR (MEMB ?SPECATTRIB DEGREEWORDS)   {...on ?SPECATTRIB}
                   (MEMB ?SPECATTRIB MUNITTYPES)
                   (MEMB ?SPECATTRIB TARGETTYPES)))
  (?INFO (OR (EQ ?INFO (QUOTE HYPOTHESES))          {...on ?INFO}
             (EQ ?INFO (QUOTE CONCLUSION]
 ([SETQ INFO (COND                                  {1st action}
      ((EQ ?INFO (QUOTE HYPOTHESES))
        (QUOTE HYPS))
      (T (QUOTE CONCLS]
  (COND                                             {2nd action}
    ((MEMB ?SPECATTRIB MUNITTYPES)
      (GETRULES (QUOTE MRULES)
                INFO ?SPECATTRIB))
    (T (GETRULES RULES INFO ?SPECATTRIB]
```

The pattern-matching interpreter invokes a pattern matcher which compares conditions against input, backtracking when necessary to explore alternatives to match variable-length pattern elements (! and !-variables). This pattern matcher is an expansion of that described in [Winston 77]. When a pattern/production rule is successfully matched, its action part is executed and the pattern-matching interpreter returns control to its caller. (...in this case, the monitor. The pattern-matching interpreter is also used by the editor.) Code for the

pattern-matching interpreter and the pattern matcher is shown below.

```
(INTERPRET
  [LAMBDA (INPUT RULEZET)
    (PROG (PALIST STARLIST MATCHING SIDECOND PTYPE TLIST TEMP ?TARGET
                  ?TYPE ?ATTRIB ?SPECATTRIB !TYPE !CMD ?VAR ?X ?INFO
                  ?FILE ?COMPAR ?INTEROG !CONCL)
          (COND
            ([for RL in RULEZET
                do (SETQ PALIST (SETQ STARLIST NIL))
                   (SETQ MATCHING (CAAR RL))
                   (SETQ SIDECOND (CDAR RL))
                   (COND
                     ((PUREMATCH MATCHING INPUT)
                       (SETVARS (LIST PALIST STARLIST))
                       (SETQ SIDECOND NIL)
                       (for ACTION in (CADR RL) do (EVAL ACTION))
                       (RETURN T]
              (RETURN T))
            (T (RETURN NIL])
```

Original MicroKnobs System Architecture
Pattern-matching System

```
(PUREMATCH
  [LAMBDA (PAT FORM)
    (PROG (P1 F1 P1CDR AP1 SCHECK)
          (COND
            ((AND (NULL PAT)
                  (NULL FORM))
              (RETURN T))
            ((NULL PAT)
              (RETURN NIL))
            ((AND (NULL FORM)
                  (NULL (CDR PAT))
                  (EQ (CAR PAT)
                      (QUOTE !)))
              (RETURN T))
            ((AND PAT (NULL FORM))
              (RETURN NIL)))
          (SETQ SCHECK NIL)
          (SETQ P1 (CAR PAT))
          (SETQ F1 (CAR FORM))
          [COND
            ((EQ (NTHCHAR P1 1)
                 (QUOTE !))
              (COND
                ((PUREMATCH (CDR PAT)
                            FORM)
                  (RETURN T))
                ((PUREMATCH (CDR PAT)
                            (CDR FORM))
                  [COND
                    ((NTHCHAR P1 2)
                      (SETQ STARLIST (CONS (CONS P1
                                                 (LIST (CAR FORM)))
                                           STARLIST]
                  (RETURN T))
                ((PUREMATCH PAT (CDR FORM))
                  [COND
                    ((NTHCHAR P1 2)
                      (SETQ STARLIST (CONS (CONS P1
                                                 (CONS (CAR FORM)
                                                       (CDAR STARLIST)))
                                           (CDR STARLIST]
                  (RETURN T))
                (T (RETURN NIL]
          [COND
            ((EQ (NTHCHAR P1 1)
                 (QUOTE ?))
```

Original MicroKnobs System Architecture
Pattern-matching System

```
            (SETQ P1CDR T)
            (SETQ AP1 (ASSOC P1 PALIST]
          (COND
            [(NOT P1CDR)
              (RETURN (COND
                        ((EQ P1 F1)
                          (PUREMATCH (CDR PAT)
                                     (CDR FORM)))
                        (T NIL]
            (P1CDR
              (COND
                [AP1 (RETURN (COND
                               ((EQ (CDR AP1)
                                    F1)
                                 (PUREMATCH (CDR PAT)
                                            (CDR FORM)))
                               (T NIL]
                [(OR (NULL SIDECOND)
                     [for SC in SIDECOND
                        do (COND
                             ((EQ (EVAL (LIST (QUOTE QUOTE)
                                              P1))
                                  (CAR SC))
                              (SETQ SCHECK T)
                              (RETURN (EVALA (CADR SC)
                                             (CONS (CONS P1 F1)
                                                   PALIST]
                     (NULL SCHECK))
                  (RETURN (PROG2 (SETQ PALIST (CONS (CONS P1 F1)
                                                    PALIST))
                                 (COND
                                   ((PUREMATCH (CDR PAT)
                                               (CDR FORM))
                                     (RETURN T))
                                   (T (SETQ PALIST (CDR PALIST))
                                      (RETURN NIL]
                (T (RETURN NIL])
```

Original MicroKnobs System Architecture
Pattern-matching System

Monitor input is interpreted first using pattern/production rules with command templates in their conditions (CM-productions), then using pattern/production rules with knowledge base query templates in their conditions (Q-productions). If none of these are successful, the monitor tells the user it was unable to recognize input. The natural language understanding of MicroKnobs is restricted to that which can be recognized with this predetermined set of templates. It is thus a pattern-oriented understanding.

The action parts of CM-productions cause system functions such as English display, choosing munitions, explaining, and editing to be invoked. The action parts of the Q-productions cause either syntactical analysis of rules in the knowledge base or knowledge base inference (in the spirit of an intelligent data base).

## 3.3. Knowledge Base and Inference

The knowledge base is stored in a set of global variables. All facts are contained in FACTS. Rules are stored in their respective rulesets. Rulesets which are to be used in inference have their names stored in RULES. The external, English form of rules and facts was shown above. Internal forms are shown below. The knowledge base display package, which is highly specialized for this application, is responsible for the transformation.

Original MicroKnobs System Architecture
Knowledge Base and Inference

```
        FACTS

1:      (THE CLOUD-COVER OF TARGET-4 IS 5)
2:      (TARGET-1 IS-A BRIDGE)
5:      (THE DEFENSE OF TARGET-1 IS LIGHTLY-DEFENDED)

        RULES

        (TRULES MRULES)

        TRULES

        (TR1 [(((?TARGET IS-A ?X)
                (FMEMB ?X (QUOTE (RADAR SAM RADIO]
              ((?TARGET IS-A EMITTER)))
        (TR2 (((?TARGET IS-A TANK-BATTALION)))
              ((THE MOBILITY OF ?TARGET IS FAST)
               (THE TEMPERATURE OF ?TARGET IS WARM)))

        MRULES

        (MR1 (((THE HARDNESS OF ?TARGET IS ?X)
                (HARDER ?X HARD))
               ((THE CLOUD-COVER OF ?TARGET IS ?Y)
                (LESS-COVERED ?Y 1))
               ((THE DEFENSE OF ?TARGET IS ?Z)
                (LESS-DEFENDED ?Z LIGHTLY-DEFENDED)))
              ((THE BEST-MUNITIONS FOR ?TARGET IS LASER-GUIDED)))
```

The inference engine is called from the monitor for choosing munitions and knowledge base querying. It uses the knowledge base, consisting of rulesets in RULES and facts in FACTS, exclusively as its domain of reasoning. It performs depth-first search and upon user-option maintains in the variable EXPLANATION lines of reasoning to answers obtained. Requests to the monitor for explanations invoke display functions which reference this variable. The inference engine includes a mechanism (not under user control) for restricting inference on hypotheses with specified key attributes (such as HARDNESS or IS-A) to members of a particular ruleset or FACTS, when the set names are stored under the RESTRICT property of the attributes. This feature allows inference to be tuned (by narrowing search) for a particular knowledge base with known interaction among rules. It can also limit flexibility in the types of rules that can be added effectively to a ruleset not named under the RESTRICT property of the added key

attributes of the consequents in the added rule. This poses a threat to users unaware of this feature.

## 3.4. Knowledge Base Editor

The knowledge base editor gives the user control over the contents of the knowledge base. He may (from the monitor) delete rules by name or facts by number, add facts to FACTS or rules to rulesets, or modify rules. He may also create new rulesets by specifying that a rule should be added to an as yet non-existent ruleset. This ruleset may then be added to RULES. Deletion of a rule or fact causes its excision from internal structure. Operations that add to the knowledge base (as well as those that modify rules) accept natural language input. Parled editor input is submitted to the pattern-matching interpreter for comparison against the component's appropriate set of pattern/production rules. Facts are compared against F-productions, hypotheses against H-productions, consequents against CQ-productions, and side-conditions (described below) against SC-productions. In each case the action part of the successful pattern/production rule establishes a variable to hold the internal (component) form resulting from the matched input. Editor functions that call the pattern-matching interpreter make use of this variable in constructing the internal forms of facts and rules. Interpreted facts are simply CONSed into FACTS. The user must specify to what ruleset an added rule is to belong, and, since location within a ruleset can affect inference, where in the ruleset it should go.

Hypotheses of MicroKnobs rules can carry side-conditions, which, like the constraints of pattern/production rules, are Lisp calls which must result in other than NIL for the hypothesis to be accepted. Side-conditions present in MicroKnobs standard knowledge base serve two purposes. Comparators (such as HARDER) can be attached as constraints to hypotheses including degreewords (e.g. HARDNESS). The internal structure

((THE HARDNESS OF ?TARGET IS ?X)(HARDER ?X HARD))

translates on output to "THE target IS HARDER THAN HARD." The expression (FMEMB ?X '(A B C)) is attached to hypotheses of the form (?TARGET IS-A ?X). This pair translates on output to "THE target IS ONE OF: A B C." While the H-productions include pattern/production rules to interpret both types of phrases whole and produce a hypothesis with attached side-condition, they also include pattern/production rules which will accept such a hypothesis without its side-condition. If the editor has reason to believe that a hypothesis it has received is not complete, it will prompt for side-conditions, which are parled, interpreted using SC-productions, and attached to the suspect hypothesis.

To modify existing rules, the user may add, delete, or replace specific hypotheses or consequents. Deleted components are removed from the rule. Components to be added are obtained by interpreting parled input, and inserted at a location specified by the user.

## 4. The LogLisp Implementation of MicroKnobs

Our goal was to preserve the outward behavior and control features of MicroKnobs while replacing the original Lisp-based implementing machinery with Logic-based machinery, as would be natural and appropriate in a LogLisp implementation.

### 4.1. Knowledge Base--Prototype Representation

The first apparent transformation opportunity was to replace MicroKnobs' inference engine and user-accessible knowledge base with Logic calls on a compatible LogLisp knowledge base. We discarded MicroKnobs's inference engine and knowledge base, and loaded the remaining system into our InterLisp version of LogLisp. We settled on an initial knowledge base representation that disposed of non-essential English in hypotheses, consequents, and facts, rearranging these components so that their key attributes appeared in the predicate position. For example, (THE HARDNESS OF ?TARGET IS ?X) became (HARDNESS target x). For ease in interfacing existing system routines, we used LogLisp's AND-special form to closely associate side-conditions with hypotheses, even though LogLisp is capable of interpreting Lisp calls without such association. Production rules with multiple consequents were distributed into separate assertions which were given names to reflect their heritage. We were able with this knowledge base to duplicate all inferences possible in the original MicroKnobs. The representative portion of our prototype knowledge base follows:

The LogLisp Implementation of MicroKnobs
Knowledge Base--Prototype Representation

```
(PROCEDURE CLOUD-COVER)

 (|- F1 (CLOUD-COVER TARGET-4 5))

(PROCEDURE IS-A)

 (|- F2 (IS-A TARGET-1 BRIDGE))

 (|- TR1 (IS-A target EMITTER)
        <- (OR (IS-A target SAM)
               (IS-A target RADIO)
               (IS-A target RADAR)))

(PROCEDURE DEFENSE)

 (|- F5 (DEFENSE TARGET-1 LIGHTLY-DEFENDED))

(PROCEDURE BEST-MUNITIONS)

 (|- MR1 (BEST-MUNITIONS target LASER-GUIDED)
        <- (AND (HARDNESS target x)
                (HARDER x HARD))
         & (AND (CLOUD-COVER target y)
                (LESS-COVERED y 1))
         & (AND (DEFENSE target z)
                (LESS-DEFENDED z LIGHTLY-DEFENDED)))

(PROCEDURE TEMPERATURE)

 (|- TR2B (TEMPERATURE target WARM)
     <-
     (IS-A target TANK-BATTALION))

(PROCEDURE MOBILITY)

 (|- TR2A (MOBILITY target FAST)
     <-
     (IS-A target TANK-BATTALION))
```

The LogLisp Implementation of MicroKnobs
Knowledge Base—Prototype Representation

We had difficulty implementing display for our prototype knowledge base. As our understanding of MicroKnobs progressed, it became apparent that this knowledge base representation would not be adequate. We had been maintaining in the ruleset variable a list of the original rule names. We displayed a ruleset by searching the entire assertion base for assertions by name, reassembling distributed rules from information contained in the assertion names. We used a similar strategy for FACTS. We modified the component output routine so that it displayed our component representation as the original had been. The resulting knowledge base display was identical to the original, but required a great deal more processing. The prototype knowledge base representation had the advantage that inference was relatively fast. The keyword predicates were more accessible than the embedded keywords of the original MicroKnobs knowledge base. Since rules of a ruleset (and subrules of a rule with multiple consequents) were distributed over Logic procedures, however, the prototype representation made it impossible to maintain control over the order in which they were examined during inference.

## 4.2. Knowledge Base—Meta-level Representation

Our desire to maintain control over assertion order led us to develop a meta-level knowledge base representation and interpreter. We now maintain the form of our prototype representation as an intact object-level knowledge base for display and editing. The new representation scheme is completed with an assertion manager that acts to ensure that the meta-level assertion base reflects the state of the object-level knowledge base before inference is attempted. Only the meta-level knowledge base is expressed in Logic.

The meta-level Logic representation transforms the prototype knowledge base by embedding the name of the ruleset (or "FACTS") around the consequent of each assertion, and embedding "TRUE" around each hypothesis. The representative portion of the meta-level knowledge base follows:

The LogLisp Implementation of MicroKnobs
Knowledge Base—Meta-level Representation

```
        (PROCEDURE MRULES)

         (|- MR1 (MRULES (BEST-MUNITIONS target LASER-GUIDED))
                 <- (AND (TRUE (HARDNESS target x))
                         (HARDER x HARD))
                  & (AND (TRUE (CLOUD-COVER target y))
                         (LESS-COVERED y 1))
                  & (AND (TRUE (DEFENSE target z))
                         (LESS-DEFENDED z LIGHTLY-DEFENDED)))

        (PROCEDURE TRULES)

         (|- TR1 (TRULES (IS-A target EMITTER))
                 <- (OR (TRUE (IS-A target SAM))
                        (TRUE (IS-A target RADIO))
                        (TRUE (IS-A target RADAR))))

         (|- TR2 (TRULES (MOBILITY target FAST))
                 <- (TRUE (IS-A target TANK-BATTALION)))

         (|- TR2 (TRULES (TEMPERATURE target WARM))
                 <- (TRUE (IS-A target TANK-BATTALION)))

        (PROCEDURE TRUE)

         (|- TRUTH (TRUE a)
             <-
             (OR (FACTS a)
                 (EVAL a)
                 (MRULES a)
                 (TRULES a)))

        (PROCEDURE FACTS)

1:       (|- (FACTS (CLOUD-COVER TARGET-4 5)))

2:       (|- (FACTS (IS-A TARGET-1 BRIDGE)))

5:       (|- (FACTS (DEFENSE TARGET-1 LIGHTLY-DEFENDED)))
```

The LogLisp Implementation of MicroKnobs
Knowledge Base—Meta-level Representation

"TRUE" is the procedure name of the meta-level interpreter, which consists of one assertion:
(TRUE a) <- (OR (FACTS a)(EVAL a)(RS1 a) ... (RSn a)), where the RSi are the rulesets in RULES, in order. The interpreter clause is generated by the assertion manager whenever necessary. The meta-level knowledge base representation and interpreter provide control over inference by maintaining assertion order within rulesets and by assuring that rulesets are explored in the order in which they appear in RULES, as in the original MicroKnobs. We also had to use an optional depth-first search mode instead of LogLisp's default non-deterministic search space traversal. A depth-first control strategy is used in the original MicroKnobs.

The meta-level Logic system is obtained from the object-level knowledge base by the assertion manager. The representative portion of the object level knowledge base follows:

The LogLisp Implementation of MicroKnobs
Knowledge Base--Meta-level Representation

```
          FACTS

(1)       (CLOUD-COVER TARGET-4 5)
(2)       (IS-A TARGET-1 BRIDGE)
(5)       (DEFENSE TARGET-1 LIGHTLY-DEFENDED)

          RULES

          (MRULES TRULES)

          MRULES

          (MR1 ((BEST-MUNITIONS target LASER-GUIDED))
               (AND (HARDNESS target x)
                    (HARDER x HARD))
               (AND (CLOUD-COVER target y)
                    (LESS-COVERED y 1))
               (AND (DEFENSE target z)
                    (LESS-DEFENDED z LIGHTLY-DEFENDED)))

          TRULES

          (TR1 ((IS-A target EMITTER))
               (OR (IS-A target SAM)
                   (IS-A target RADIO)
                   (IS-A target RADAR)))
          (TR2 ((MOBILITY target FAST)
                (TEMPERATURE target WARM))
               (IS-A target TANK-BATTALION))
```

The assertion manager is embedded around calls to inference. It reasserts changed procedures before attempting inference. It checks the contents of two new global variables, FACTSCHANGED and CHANGEDRULESETS to see whether the knowledge base has been changed. We modified the editor so that any time a fact is added or deleted, FACTSCHANGED is set to T. Any time a ruleset is edited, the ruleset name is added to CHANGEDRULESETS if it is not already there. After erasing and reasserting (in meta-level) changed procedures, the assertion manager sets both variables back to NIL and perfoms the required inference.

The assertion manager reasserts changed procedures by inserting calls to TRUE in the proper locations in OR- and AND-special forms. Side-conditions are identifiable as the second half of an AND-special form.

## 4.3. Knowledge Base--Restricted Meta-level Representation

A refinement in the meta-level representation resulted from the observation that inference restrictions as they exist in the original MicroKnobs could be easily implemented if hypothesis translation code in the assertion manager were not so eager to have the interpreter called at every juncture. We rewrote this code to translate a hypothesis into an OR-special form of only procedure calls to which its input had been restricted. This has an effect identical to that of restriction in the original MicroKnobs.

The representative portion of the restricted meta-level knowledge base follows:

The LogLisp Implementation of MicroKnobs
Knowledge Base--Restricted Meta-level Representation

```
(PROCEDURE MRULES)

(|- MR1 (MRULES (BEST-MUNITIONS target LASER-GUIDED))
        <- (AND (TRULES (HARDNESS target x))
                (LISP (HARDER x HARD)))
         & (AND (FACTS (CLOUD-COVER target y))
                (LISP (LESS-COVERED y 1)))
         & (AND (OR (FACTS (DEFENSE target z))
                    (TRULES (DEFENSE target z)))
                (LISP (LESS-DEFENDED z LIGHTLY-DEFENDED))))

(PROCEDURE TRULES)

(|- TR1 (TRULES (IS-A target EMITTER))
        <- [OR (OR (FACTS (IS-A target SAM))
                   (TRULES (IS-A target SAM)))
               (OR (FACTS (IS-A target RADIO))
                   (TRULES (IS-A target RADIO)))
               (OR (FACTS (IS-A target RADAR))
                   (TRULES (IS-A target RADAR])

(|- TR2 (TRULES (MOBILITY target FAST))
        <- (OR (FACTS (IS-A target TANK-BATTALION))
               (TRULES (IS-A target TANK-BATTALION))))

(|- TR2 (TRULES (TEMPERATURE target WARM))
        <- (OR (FACTS (IS-A target TANK-BATTALION))
               (TRULES (IS-A target TANK-BATTALION))))

(PROCEDURE TRUE)

(|- TRUTH (TRUE a)
    <-
    (OR (FACTS a)
        (EVAL a)
        (MRULES a)
        (TRULES a)))

(PROCEDURE FACTS)

1:  (|- (FACTS (CLOUD-COVER TARGET-4 5)))

2:  (|- (FACTS (IS-A TARGET-1 BRIDGE)))

5:  (|- (FACTS (DEFENSE TARGET-1 LIGHTLY-DEFENDED)))
```

## The LogLisp Implementation of MicroKnobs Knowledge Base—Restricted Meta-level Representation

After settling on a knowledge base representation, we made the appropriate changes to the H-productions and CQ-productions. We eliminated the SC-productions, since side-conditions are theoretically unnecessary, since we had replaced the FMEMB-side-conditions with OR-special forms, and since we could not get pattern/production rules in the SC-productions that dealt with comparators to work, even in the original MicroKnobs. We changed the editor to operate on our object-level knowledge representation, and eliminated from it all code which pertained to side-conditions. We did not eliminate such code from the display package, since comparator side-conditions are still attached to hypotheses by some of the H-productions. We did eliminate from the H-productions the pattern/production rules which accepted bare hypotheses that would not make sense without side-conditions.

We modified CHOOSE-MUNITIONS (the function used to choose munitions, called from CM-productions) and the inferential knowledge base queries in Q-productions to call the assertion manager where they originally called the MicroKnobs inference engine. We wrote code that produces a data structure compatible with the MicroKnobs routine for displaying explanations by extracting the appropriate information from #DERIVATIONS, the LogLisp system global variable accessed by LogLisp's explanation facility.

In retrospect, it was unnecessary to change the component-level representation (hypotheses, consequents, facts) from the original pseudo-English of MicroKnobs. The meta-level representation we now use for the knowledge base does not require key attributes to come first, and, since terms are not indexed, it affords no greater efficiency. The original component representation admits some flexibility in interpreting new rules that ours does not. If a component "falls through" its set of pattern/production rules it is accepted by MicroKnobs as is. Our implementation also accepts the component as is, except that it then has a form different from other, similar components in the knowledge base. The new component cannot be excluded from valid inference, however, because it will match with no part of the knowledge base which did not also fall through its set of pattern/production rules when interpreted, provided that the rules are complete for the forms they accept. Our choice of component representation therefore caused us no loss of flexibility; it only made us work unnecessarily hard. Moreover, we are embarrassed every time a component falls through and gets the representation that we should have given it.

## 4.4. Pattern-matching System

We analyzed the software architecture of MicroKnobs to identify other system components which might be comfortably expressed in Logic, and decided on the pattern-matching system and the dictionary. The Lisp code for the MicroKnobs pattern matcher (listed in the previous section) is very complex and probably would not have been written by a LogLisp programmer. The pattern-matching interpreter employs a number of PROG variables global to the pattern-matching process, including names for pattern variables and association lists (PALIST and STARLIST) to hold their values. Such devices are not required for a pattern matcher implemented in Logic (especially with the addition of resolution semantics for sequential Logic forms [Schrag 84]), since unification pattern-matching and its intrinsic variable access mechanisms are fundamental to logic programming. Unification forms the foundation of a Logic-coded MicroKnobs pattern matcher. If not for variable-length pattern elements (! and !-variables), no additional pattern-matching code would be necessary. The logic programming feature of backtracking is ideal for implementing the non-deterministic search required by patterns containing such elements. The Logic-coded pattern matcher shown below is based on a model of transformations [Emanuelson 82] of pattern and input.

The LogLisp Implementation of MicroKnobs
Pattern-matching System

```
                                                          {(comments)}
(PROCEDURE Trans)

 (|- Basis (Trans (Pat)                                   {basis case
                  (Inp)))                                    for recursion}

 (|- !=() (Trans (Pat ! . ptail)                          {match ! to null}
                 (Inp . itail))
          <- (Trans (Pat . ptail)
                    (Inp . itail)))

 (|- !=# (Trans (Pat ! . ptail)                           {match ! to any
                (Inp # . itail))                                element}
         <- (Trans (Pat ! . ptail)
                   (Inp . itail)))

 (|- !<-nil (Trans (Pat (! NIL) . ptail)                  {terminate the
                   (Inp . itail))                           list bound to
            <- (Trans (Pat . ptail)                          a !-variable}
                      (Inp . itail)))

 (|- !<-propid (Trans (Pat (! (unit . seg)) . ptail)      {cons unit into
                      (Inp unit . itail))                   the list bound
               <- (Trans (Pat (! seg) . ptail)              to a !-variable}
                         (Inp . itail)))

 (|- ?<-propid (Trans (Pat (? unit) . ptail)              {bind unit to
                      (Inp unit . itail))                       a ?-variable}
               <- (Trans (Pat . ptail)
                         (Inp . itail)))

 (|- ?<*propid (Trans (Pat (? unit constraint) . ptail)
                      (Inp unit . itail))                 {bind unit to a
               <- constraint                               ?-variable if it
                  (Trans (Pat . ptail)                        satisfies
                         (Inp . itail)))                        constraint}

 (|- Propid=propid (Trans (Pat propid . ptail)            {match two
                          (Inp propid . itail))            instances of the
                   <- (Trans (Pat . ptail)                    same proper
                             (Inp . itail)))                    identifier}
```

Pat and Inp are preface keywords that prevent fortuitous reduction of pattern and input expressions when they begin with atoms possessing Lisp function definitions. !- and ?-variables are represented in list form (e.g. (! var-name)) to facilitate processing. Constraints for ?-variables are also handled well with this representation.

A pattern/production rule (of Logic Q-production procedure QPRD) corresponding to the one listed in Section 3 and compatible with the Logic pattern matcher procedure Trans is listed below.

```
(|- (QPRD input)
                <- (Trans [Pat [? interog (OR (EQ interog
                                                  (QUOTE WHAT))
                                              (EQ interog
                                                  (QUOTE WHICH]
                               !
                               [? attrib (OR (EQ attrib (QUOTE CHOOSE))
                                             (EQ attrib (QUOTE
                                                          MENTION]
                               !
                               [? specattrib
                                  (OR (MEMB specattrib (EVAL
                                                         DEGREEWORDS))
                                      (MEMB specattrib (EVAL
                                                         MUNITTYPES))
                                      (MEMB specattrib (EVAL
                                                         TARGETTYPES]
                               !
                               (? info (OR (EQ info (QUOTE HYPOTHESES)
                                                    )
                                           (EQ info (QUOTE CONCLUSION]
                           input)
                      [SEQUENCE [= part (COND ((EQ info (QUOTE
                                                          HYPOTHESES))
                                               (QUOTE HYPS))
                                              (T (QUOTE CONCLS]
                                (COND ((MEMB specattrib (EVAL MUNITTYPES)
                                                        )
                                       (GETRULES (QUOTE MRULES)
                                                 part specattrib))
                                      (T (GETRULES RULES part specattrib])
```

The major difference between this Logic rule and its Lisp counterpart is that the Logic code is executable, embodying active control, whereas the Lisp code represents a more passive data structure. SEQUENCE is a sequential Logic form with resolution semantics described in [Schrag 84]. It succeeds if and only if all of its Logic subforms succeed and all of its Lisp subforms are evaluable.

The interpreter for this pattern-matching system is the simple function shown below.

```
(INTERPRET
  [NLAMBDA (INPUT RULEZET)
    (SETOF 1 T (LIST (LIST RULEZET (CONS (QUOTE Inp)
                                         INPUT])
```

This function merely packages a Logic call to invoke the pattern-matching system. The keyword Inp again prefaces input so that it is not fortuitously reduced.

## 4.5. Dictionary

Our Logic implementation of the MicroKnobs dictionary of course uses assertions rather than property lists for the storage and association of preferred terms with synonyms. The assertions for the preferred term AIRBASE and the compatible version of GETROOTWORD are shown below.

```
(PROCEDURE SYNONYM)

 (|- (SYNONYM AIRBASE AIRFIELD))

 (|- (SYNONYM AIRBASE AIRPORT))

 (|- (SYNONYM AIRBASE TUOC))

 (|- (SYNONYM AIRBASE (AIR BASE)))

 (|- (SYNONYM AIRBASE (AIR FIELD)))
```

The LogLisp Implementation of MicroKnobs Dictionary

```
(GETROOTWORD
  [LAMBDA (WORD)
    (SETOF 1 (QUOTE x)
           (LIST (LIST (QUOTE SYNONYM)
                       (QUOTE x)
                       WORD])
```

There is one assertion for each synonym of a preferred term. GETROOTWORD invokes Logic search over the dictionary's SYNONYM procedure.

## 5. The Baseline Investigation

One must take care when comparing a program designed in one language to its reimplementation in another. There are bound to have been design decisions based on advantages and constraints of the original language which the target language does not share. Nonetheless we must compare our LogLisp implementation with the original MicroKnobs in terms of programming ease, program clarity, and program efficiency. We shall temper our comparison with an awareness of language differences.

### 5.1. Programming Ease

The most difficult facet of implementing MicroKnobs in LogLisp was for us to understand the original code: how it behaved, how it worked, and which of its workings we needed to preserve its behavior. MicroKnobs includes about 100 Lisp functions and several large variables. We acquired a working knowledge of most of these, and left much of the code intact.

Our task was made easier by the quality of the original code. The division of labor among functions seems natural. Functions and variables for a particular architectural component are grouped together on one InterLisp file, except where there is overlap. The names of functions and variables are mnemonic, making the code easy to follow once we understood the mnemonic dialect. One thing that made the transition particularly smooth is the judicious use (to which we did not always adhere) of "macros" (the interpreted versions were actually used) for accessing data structures. When we changed the structure of the knowledge base, we simply redefined the macros. While this did not solve all of our problems, it was certainly much easier than it would have been translating primitives, if there had been no macros.

The major facility encountered in programming in LogLisp rather than plain InterLisp is that we did not have to write an inference engine; LogLisp supplied one. On the other hand, our desire to preserve the behavior of MicroKnobs and subsequent object-level/meta-level knowledge base representation led to the need for an assertion manager. We do not fully comprehend the partitioning of the knowledge base into rulesets (and FACTS) that exists in MicroKnobs. We would have far preferred a predicate-based partitioning, and would have used one had it not made a faithful reimplementation impossible. The discovery of an operable meta-level strategy was essential to our effort, but resulted in losses of efficiency and clarity over conventional methods. We do not think the burden of writing an assertion manager is an indictment of LogLisp's suitability for the development of knowledge-based systems,

though, only for their reimplementation.

Similarly, a significant portion of the necessary pattern-matching capability was provided by Logic. In particular, pattern-matching variables and their bindings are handled transparently by Logic. Design of the Logic-coded pattern matcher was straightforward, since the input/output relationships specified by its component clauses correspond naturally to the required transformations.

Design of the dictionary function is not very complex in either implementation.

In the end, we took a handful of functions out of MicroKnobs, and put a handful back in.

## 5.2. Program Clarity

We feel that the LogLisp version has enhanced program clarity. While much code is shared by both versions, our object-level representation of the knowledge base is easier to read. Lisp calls appear more natural with predicate status than as attached side-conditions to hypotheses. Our rule format, which follows the syntax of a LogLisp assertion with a list of consequents in the normal position of the single consequent, also seems natural. The meta-level knowledge base and associated knowledge base manager are perhaps less transparent.

We feel that the Logic-coded pattern-matching system is more concise and perspicuous than its Lisp-coded counterpart. This is certainly evidenced by the easy job we had programming it. It may be easier to read template portions in the original pattern/production rule representation (where they are uninterrupted), but the overall rule structure seems more homogeneous (and more easily apprehended) in our representation.

Again, in both implementations the dictionary system is so simple that clarity is not really an issue.

## 5.3. Program Efficiency

The changes we made to the original version that significantly affected efficiency involved code for inference (including assertion management and explanation), pattern-matching interpretation, and dictionary processing. The measurements discussed here were made using a compiled instance of LogLisp V2M3. MicroKnobs modules or functions were always left in intrepreted InterLisp, except where, in the

conventional Lisp implementation, they performed duties corresponding to those for which the efficiency of the LogLisp implementation was to be compared. Thus, fair measurements were made of MicroKnobs original inference, matching, and dictionary processing. While dynamic storage (consing) costs are not reported here, their differences were generally proportional with the reported execution time cost differences.

Table 1 shows measurements in CPU seconds of inference execution time for a battery of questions in various implementations of MicroKnobs. The conventional Lisp implementation was measured both with and without restrictions on knowledge base search. LogLisp implementations with our ultimate restricted meta-level knowledge representation, unrestricted meta-level representation, and prototype representation were also measured. Because restriction in MicroKnobs is predicate-based, our prototype knowledge base is also inherently "restricted."

| Question | Unrestr Lisp | (Orig) Restr Lisp | Restr Meta Logic | UnRestr Meta Logic | Proto Logic |
|---|---|---|---|---|---|
| CHOOSE FOR TARGET 1. | .166 | .125 | .399 | .576 | .270 |
| CHOOSE FOR TARGET 2. | .335 | .257 | .670 | 1.206 | .322 |
| CHOOSE FOR TARGET 3. | .459 | .367 | .871 | 1.571 | .373 |
| CHOOSE FOR TARGET 4. | .671 | .528 | 1.388 | 2.643 | .569 |
| WHAT IS TARGET 3? | .009 | .009 | .066 | .073 | .061 |
| IS TARGET 3 AN SA-4? | .008 | .010 | .057 | .071 | .027 |
| IS TARGET 3 A SAM? | .041 | .044 | .240 | .292 | .088 |
| IS TARGET 3 AN EMITTER? | .078 | .087 | .315 | .414 | .128 |
| HOW HARD IS TARGET 1? | .034 | .029 | .178 | .211 | .062 |
| HOW HARD IS TARGET 2? | .109 | .084 | .243 | .369 | .071 |
| HOW HARD IS TARGET 3? | .108 | .095 | .244 | .352 | .066 |
| HOW HARD IS TARGET 4? | .079 | .072 | .217 | .303 | .074 |
| Total: | 2.097 | 1.707 | 4.888 | 8.131 | 2.111 |

Table 1

Table 1 contains two items of information especially useful for baseline comparison. At a gross level, when data from the original implementation is used as baseline, our meta-level implementation is about five times slower; restricted meta-level, about three times slower; and prototype, about the same. It can also be seen at a gross level that restriction confers a greater degree of efficiency enhancement in the original InterLisp implementation. The relative

impacts of the differing designs of components used in inference in the conventional InterLisp and LogLisp implementations can be assessed from Table 2, which shows measurements of execution times for question 2, "CHOOSE MUNITIONS FOR TARGET 4," for which there is no answer and exhaustive search is performed.

| Function | Unrestr Lisp | (Orig) Restr Lisp | Restr Meta Logic | UnRestr Meta Logic | Proto Logic |
|---|---|---|---|---|---|
| search | .268 | .190 | .250 | .423 | .119 |
| reduction | .074 | .079 | .416 | .899 | .176 |
| unification | .295 | .211 | .444 | .740 | .125 |
| environment management | .034 | .037 | .139 | .264 | .097 |
| index identification | -- | -- | .111 | .264 | .040 |

Table 2

The components have been identified so as to correspond as closely as possible within the given designs. Search is defined to include search space traversal and management, knowledge base access, and explanation maintenance. Reduction is the instantiation and evaluation of Lisp constraints. Unification is elementary pattern matching. Environment management pertains to the bindings of inference variables, and index identification to extraction of constants from goal statements to enable indexed retrieval of *assertions*. (*The last component is* present only in the LogLisp implementations.)

Restriction confers an efficiency enhancement for both search and unification of approximately 30% in the original InterLisp implementation, and of approximately 40% in the LogLisp meta-level implementations. The additional enhancement in LogLisp corresponds to elimination of the TRUTH procedure and its processing. Environment management cost is essentially constant in the conventional InterLisp implementations, but grows with search space size in LogLisp. This is to be expected, as LogLisp provides binding capability for every processed goal statement, whereas the original design merely keeps a list of explicitly encountered variables and bindings. Time required for reduction is essentially constant in the conventional InterLisp implementations. In LogLisp it grows with search space and knowledge representation size, at roughly the same rate as that for index identification. This is to be expected, as LogLisp attempts to reduce every s-expression it encounters in a goal statement, whereas the original knowledge representation isolates Lisp constraints in a predetermined location.

Assertion management confers a modicum of overhead not present in the original MicroKnobs. Because of the limited size of the knowledge base this is not excessive, amounting in the worst case (all procedures erased and reasserted) to about .4 CPU seconds. With a more realistic (larger) knowledge base, this overhead would become intolerable, and a different approach to assertion management would be necessary. LogLisp V2M3 includes a solution to this problem in its capability for modification of single assertions by name—rather than only of entire procedures—but was not available when the assertion manager was designed.

The generation of a compatible data structure for the display of explanation is not expensive either, taking up to 1.5 seconds for tested queries.

Table 3 shows measurements in CPU seconds of pattern-matching interpretation execution time for selected input forms using either the CM-production or Q-production pattern/production ruleset, exclusively.

| Question | Pattern Ruleset | Lisp | Logic |
|---|---|---|---|
| CHOOSE MUNITIONS FOR TARGET-1 | CM-prod's | .479 | 1.256 |
| WHAT IS TARGET-3 | Q-prod's | .332 | 3.600 |
| WHAT IS THE HARDNESS OF TARGET-1 | Q-prod's | .447 | 6.211 |
| THIS OLD FOOL HAS NO HOME | CM-prod's | .121 | 3.914 |
| THIS OLD FOOL HAS NO HOME | Q-prod's | .216 | 1.903 |

Table 3

These execution times are averaged over two or three trials, and include the cost of inference and other "action" times in addition to that of pattern matching. The nonsense query, "THIS OLD FOOL HAS NO HOME," has no matching template among either ruleset, and should be more indicative of the proportionate cost of pattern-matching processing than the other input forms. The figures show a great degree of variability, however, preventing the drawing of very specific conclusions about their meanings. The Lisp-coded pattern matcher certainly specializes more in its invocation of recursive calls than the Logic-coded pattern matcher, and uses low-level details inaccessible to logic programmers such as the binding condition of a pattern variable (whether instantiated).

Because of LogLisp's full secondary indexing for ground assertions, access of preferred words for synonyms in dictionary processing occurs in essentially constant time in the Logic version, averaging about .37 seconds over all known pairs. In the original InterLisp version, where simple sequential search is performed over the list of preferred words, access time increases linearly with the depth of the target word in the list, averaging (for this relatively small dictionary) about .0003 seconds. LogLisp might become competitive with a dictionary two orders of magnitude larger, if the storage required for this cross-referencing were not prohibitive.

In general, these are fairly predictable flexibility/efficiency trade-offs. User-specifiable index fields for rule assertions available in LogLisp Version 3 [Robinson et al. 84] should ameliorate the overhead problems of the data structures used in the Logic-coded pattern-matcher and meta-level knowledge representations.

## 5.4. Results

The results indicate that LogLisp is a viable environment for the implementation and development of knowledge-based systems, especially if it can be used naturally. Even when it is used in a forced manner as we have, the advantages it affords in ease of programming, clarity, and flexibility outweigh the disadvantage in efficiency. We have found LogLisp to be effective in the implementation of a small, simple knowledge-based system. It is now appropriate to consider the implementation of a non-trivial AI program in LogLisp to demonstrate its effectiveness in a practical application.

## 6. References


[Emanuelson 82] P. Emanuelson. "From Abstract Model to Efficient Compilation of Patterns," Linkoping University research report LITH-MAT-R-82-03.

[Engelman 79] C. Engelman. MicroKnobs Documentation, MITRE Corporation.

[Engelman 81] C. Engelman. "KNOBS, An Interactive Knowledge Based Tactical Air Mission Planning Demonstration System: A Snapshot as of September 1981," MITRE Corporation.

[Kowalski 79] R. Kowalski. Logic for Problem Solving, Elsevier North-Holland.

[Robinson 81] J. A. Robinson. "VHL System Breadboard: Annual Progress Report," Syracuse University.

[Robinson and Sibert 81] J.A. Robinson and E.E. Sibert. The LogLisp User's Manual, unpublished interim technical report.

[Robinson et al. 84] J.A. Robinson, E.E. Sibert, and K.J. Greene. The LogLisp Programming System, RADC technical report F30602-81-C-0024

[Schrag 83] R.C. Schrag. "Notes on the Conversion of LogLisp from Rutgers/UCI-Lisp to InterLisp," RADC technical memorandum RADC-TM-83-1.

[Schrag 84] R.C. Schrag. "LogLisp Sequential Forms with Resolution Semantics," RADC technical memorandum RADC-TM-84-13, Jun 84

[Teitelman 78] W. Teitelman. INTERLISP Reference Manual, Bolt, Beranek, and Newman and Xerox Corporation.

[Winston 77] P.H. Winston. Artificial Intelligence, Addison-Wesley.

[Winston and Horn 80] P. H. Winston and B.K.P. Horn. Lisp, Addison-Wesley.

# *MISSION*
# *of*
# *Rome Air Development Center*

*RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence ($C^3I$) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.*

END

FILMED

9-84

DTIC